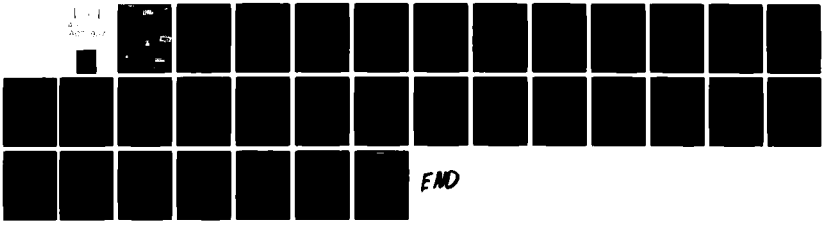
AD-A079 987 MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB F/G 9/3
PARALLEL OPERATIONS ON BINARY IMAGES.(U)
NOV 79 R KLETTE AFOSR-77-3271
UNCLASSIFIED TR-822 AFOSR-TR-80-0078 NL



TR-[illegible] 80-078

LEVEL II

AD A079987

DDC
DECEMBER
JAN 29 1980
RECEIVED

DDC FILE COPY

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

This document has been approved for public release and sale; its distribution is unlimited.

80 1 29 012

(12)

(9) Interim report

(14) TR-822

(15) AFOSR-77-3271

(11) November 79

(12) 33

(6) PARALLEL OPERATIONS ON BINARY IMAGES.

(10) Reinhard Klette*
Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

(14) AFOSR

(17) TR-80-0076

(16) 2304

(17) A2

ABSTRACT

It is a well-known fact that parallel logical operations and shifts are useful for speeding up certain computational tasks in binary image processing. A theoretical model for computation is given using these operations as basic instructions. Some examples demonstrate the utility of such a parallel processing system for fast solutions, e.g., the recognition of rectangles, squares, and isosceles right triangles can be done within time O(log N) for input images of size NxN.

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Kathryn Riley. The author also thanks Prof. Azriel Rosenfeld for his careful reading of the manuscript.

*Permanent address: Sektion Mathematik, Friedrich-Schiller-Universität, DDR-69 Jena, Universitätshochhaus 17.OG, German Democratic Republic.

411 074

JOB

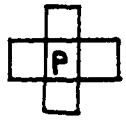
1. Introduction

In this paper, a binary image is a pattern of 1's on a background of 0's, filling out a scanning field of a certain size NxM. In the binary image

$$X = (X_{ij}) \quad i=0,1,\ldots,N-1;\ j=0,\ldots,M-1$$

$X_{ij}=0$ means that (i,j) is a background point, and $X_{ij}=1$ that (i,j) is an object point. It is well known (see, e.g., [1]) that many operations on binary images such as edge detection, shrinking, expanding, or projection can be performed in parallel. The following simple example of edge detection demonstrates this computational approach:

Assume that all points (i,j) with $X_{ij}=1$ represent a certain object obtained as a result of thresholding a gray level image of this object. The contour of the object is defined to be the set of all points (i,j) with $X_{ij}=1$, such that in the neighborhood of (i,j) there is at least one point (k,ℓ) with $X_{k\ell}=0$. Let us use the 4-neighborhood



for points p = (i,j) in such a binary image. Then, for a binary image X the following procedure computes an edge image X* in which $X^*_{ij}=1$ iff the point (i,j) is a contour point in X (cp. [1,2]):

1. Y ← shift X one column to right;
2. Z ← AND(X,Y) for all points of X and Y in parallel;

3. Y ← shift X one column to left;

4. Z ← AND(Z,Y) for all points of Z and Y in parallel;

5. Y ← shift X one row up;

6. Z ← AND(Z,Y) for all points of Z and Y in parallel;

7. Y ← shift X one row down;

8. Z ← AND(Z,Y) for all points of Z and Y in parallel;

9. Z ← NON(Z) for all points of Z in parallel;

10. X*← AND(X,Z) for all points of X and Z in parallel.

Thus, the computation of X* = edge(X) can be done within 10 steps, independently of the size NxM of the binary image X, where in each step a logical operation or a shift will be performed on the whole image field in parallel. On the other hand, using a sequential computer, the computation of X* requires O(NxM) computational steps.

Using CCD or optical techniques, such a parallel procedure can be implemented in hardware [3,4]. However, our considerations are from a theoretical standpoint and are not concerned with questions of implementation.

The paper is organized as follows: In Section 2 the computation model for parallel operations on binary images is defined, following the definition in [2]. In Sections 3, 4, and 5 we discuss some examples of binary image processing demonstrating the utility of the available parallelism for fast solutions. Section 3 discusses local operations; Section 4 the recognition of rectangles, squares, and isosceles right triangles, and

Section 5 the computation of area and perimeter for a simply connected object [1].

2. Definition of the PBS

For reasons of simplicity we will consider binary images of size $N=M=2^p$, for integer $p \geq 2$. This choice is motivated by the fact that a power of 2 is optimal for the image size in some sense: Binary image transforms via the image processing computer used in this paper require in many cases the same time complexity for all sizes N, $2^p < N \leq 2^{p+1}$, for any given positive integer p.

Let $\mathcal{B}$ be the set of all binary images of size NxN. For $X \in \mathcal{B}$, X_{ij} or $X(i,j)$ denotes the value of the binary image X at the point (i,j). Points (i,j) outside the scanning field $\{0,1,\ldots,N-1\}^2$ are always assumed to have value 0.

Our model for computation represents a hybrid system of a vector machine [5] with a matrix machine [6]. Generally speaking, the matrix machine computes binary image transforms, and the operation of this two-dimensional machine is controlled by switching in a vector machine. Both machines are parallel processors working as single-instruction-multiple-data (SIMD) computers. We denote our model for computation by PBS (paralleles Binärbildverarbeitungssystem).[1]

The PBS potentially makes use of countably many registers of three types:

---

[1] In [2] this system was called IPC, but we think that the denotation PBS is more appropriate. In fact, the definition of the PBS fully agrees with the IPC definition given in [2].

(i) vector registers A,B,C,...

(ii) index registers I,J,K,...

(iii) matrix registers X,Y,Z,...

All registers may be subscripted.

Vector registers store one vector of the set $V:=\{0,1\}^N$ at a time. For simulating operations on nonnegative integers, we adopt the traditional convention of writing the sequence of bits (elements of $\{0,1\}$) from right to left. Hence, for the actual value $\langle A\rangle$ of a vector register A the relation $0\leq\langle A\rangle\leq 2^N-1$ is always true. Furthermore, we can use vector registers for simulating operations on negative integers or on real numbers. In such cases, we need special conventions for the meaning of some bit positions. Note that the length N of a stored vector is identical to the size N of the binary images.

Index registers store one vector of the set $0^*\cup 0^*10^*$ of length log N+1 at a time. Hence, for the actual value $\langle I\rangle$ of an index register I we have $\langle I\rangle\in\{0,1,2,4,\ldots,N\}$ at any time. Usually these auxiliary registers will be used for storing shift distances, or for loop counting.

Matrix registers store one binary image $X\in B$ at a time. That is, each register of this type stores exactly N^2 bits in a two-dimensional square array. For example, in the uppermost row of such a register X the N bits

$$X_{00}\ X_{01}\ X_{02}\cdots X_{0,N-1}$$

are stored in this order.

The instructions for a PBS program are drawn from the instruction set given in Table 1. In what follows, we will consider each instruction to have a cost of one unit of time (uniform cost criterion). The time of a computation is its length (number of executed basic instructions). For example, we may utilize while-loops, or other bitwise parallel Boolean operations such as $\vee$ and $\oplus$. However, it should be clear how to translate such constructs into the austere language described in Table 1 at a cost of a constant factor in time.

Note that our instructions guarantee that the contents of a vector register are never assigned to an index register and vice versa. Any index register contains at most one 1 at any time.

We write programs in an Algol-like notation, cp. [7]. Because it is not necessary that our language should be implemented, we take liberties with the programming language.

A program for the PBS is a sequence of instructions. Our programs are deterministic. On the top of a program we enumerate those registers which will be used during the computation. At the start of a program all non-input registers are identically 0.

For example, the program for computing an edge image (cp. the example in the Introduction) has the form

```
begin index I; image X,Y,Z; read X; int I=1;
      Y:=X→I; Z:=X∧Y; Y:=X←I; Z:=Z∧Y;
      Y:=X↑I; Z:=Z∧Y; Y:X↓I; Z:=Z∧Y;
      Z:=Z̄; Y:=X∧Z; print Y
end
```

Instruction	Function
int A=m; int I=2^n; image X=X_0;	constant vectors or binary images into registers A,I,X, esp. <A>=m, <I>=2^n, and X=$X_0 \in \mathcal{B}$;
read A; read X;	input statements for vector and matrix registers;
print A; print X;	output statements for vector and matrix registers;
A:=B; I:=J; X:=Y;	direct assignments;
A:=$\bar{B}$; A:=B$\wedge$C; X:=$\bar{Y}$; X:=Y$\wedge$Z;	bitwise parallel Boolean operations,
A:=B$\leftarrow$I; A:=B$\rightarrow$I; I:J$\leftarrow$; I:=J$\rightarrow$; X:=Y$\leftarrow$I; X:=Y$\rightarrow$I; X:=Y$\uparrow$I; X:=Y$\downarrow$I;	shift operations, shift distances given by <I> for vector and matrix register, for index registers the shift distance is restricted to one bit position, the bits shifted out are discarded, the vacated positions are filled with 0's;
if A$\{\overset{=}{\neq}\}$0 then...else...fi;	test instructions for vector and index registers;
if I$\{\overset{=}{\neq}\}$0 then...else...fi;	
X:=A;	A *into the lowest row of* X, the vacated positions in X are filled with 0's;
A:=X;	the lowest row of X into A.

Table 1. Basic PBS Instructions.

According to a given program the PBS computes a certain function

$$f: \mathcal{V}^{n_1} \times \mathcal{B}^{n_2} \to \mathcal{V}^{m_1} \times \mathcal{B}^{m_2}.$$

Possible interpretations of such functions are listed in Table 2. Such a function belongs to the complexity class

PBS - TIME(T(N))

iff this function can be computed on PBS within time O(T(N)). In this, the time complexity is taken as the maximum complexity over all inputs (worst-case complexity). E.g., we have seen that the function edge:$\mathcal{B}\to\mathcal{B}$ belongs to the complexity class PBS-TIME(1).

According to Table 1, test instructions for matrix registers (Is X identical to 0 in all bit positions, or not?) may be performed by a PBS subroutine within time O(log N). To see this, let

zero(X):=if X≡0 then 1 else 0 fi.

The function zero:$\mathcal{B}\to\mathcal{V}$ belongs to the complexity class PBS-TIME(log N):

```
begin vector A; index I; image X,Z;
      int I=N/2; read X;
      while I≠0 do
          Z:=X↓I; X:=X∨Z; I:-I→ od;
      A:=X;
      if A=0 then print 1 else print 0 fi;
end
```

Function	Fields of Application
$f: B^n \rightarrow B$	image processing
$f: B^n \rightarrow V$	feature extraction, image coding, recognition
$f: V^n \rightarrow V$	classification, auxiliary operations
$f: V^n \rightarrow B$	image reconstruction, image decoding

Table 2. Some Function Interpretations.

The principle of this program is easy to understand. This example demonstrates the utility of power-of-2 shifts using a divide-and-conquer approach.

In the next sections we will consider some examples of binary image transforms. These examples are selected for presentation without any special motives. We want to emphasize the utility of the available parallelism for image processing tasks. Some examples of binary image processing using the PBS can be found in [2], e.g.:

1. Let number(X) be the quantity of 1's in the binary image X. This function number:$\mathcal{B}\rightarrow\mathcal{V}$ belongs to the class PBS-TIME $((\log N)^2)$.

2. Let proj(X) be the projection of X in the column direction, i.e., all 1's in X are shifted down to the lowest row. This function proj:$\mathcal{B}\rightarrow\mathcal{B}$ can be computed on PBS within time O(N).

3. Local operations

Let $f:\mathcal{B}\to\mathcal{B}$ be a local operator on binary images, i.e., f is defined using a particular neighborhood, and a particular logical function on the points in this neighborhood. For example, the operator edge is defined using the neighborhood

$$\begin{array}{ccc} & x_2 & \\ x_3 & x_0 & x_1 \\ & x_4 & \end{array}$$

and the Boolean function

$$x_0 \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) = x_0 \wedge non(x_1 \wedge x_2 \wedge x_3 \wedge x_4).$$

In general [2], an operator $f:\mathcal{B}\to\mathcal{B}$ is called a local operator of degree n iff there exists a totally ordered "shifting set" E with card E=n, containing integer tuples, and there exists a Boolean function $f^*:\{0,1\}^n\to\{0,1\}$, with the following property:

The set E can be represented by the n-tuple $[(i_1,j_1), (i_2,j_2),\ldots,(i_n,j_n)]$, according to the total order on E. Then, via

$$f(X)(i,j) = \underline{\text{if}}\ (i,j)\in\{0,1,2,\ldots,N-1\}^2\ \underline{\text{then}}\ f^*(X_{i+i_1,j+j_1}, X_{i+i_2,j+j_2},\ldots,X_{i+i_n,j+j_n})\ \underline{\text{else}}\ 0\ \underline{\text{fi}};$$

the operator f can be computed by using the Boolean function f^* in the environment E.

Let LOC_n be the class of all local operators of degree n, for $n\geq 0$, and let $LOC := \bigcup_{n=0}^{\infty} LOC_n$ be the class of all local operators. In [2] it was shown that

1. if $n<(2N-1)^2$ $\quad LOC_n \subset LOC_{n+1}$;
2. $LOC_n=LOC$, for all $n\geq(2N-1)^2$; and
3. the cardinality of the class LOC is equal to

$$2^{2^{(N-1)^2}\cdot[N\cdot 2^N\cdot(N\cdot 2^{N-1}-N+1)+(N-1)^2]}$$

From property 3 we have in particular that there exist operators $f:\mathcal{B}\to\mathcal{B}$ which are not in LOC.

Let $f\in LOC$ be an arbitrary operator with environment $[(i_1,j_1),(i_2,j_2),\ldots,(i_n,j_n)]$ and with Boolean function f^*. Then f can be computed on PBS according to the following general program:

```
begin image X,Z_1,Z_2,...,Z_n; read X;
      for t:=1 step 1 until n do
          Z_t:=(X←j_t)↑i_t od;
      X:=f*(Z_1,Z_2,...,Z_n);
      print X
end
```

where for negative integers j_t or i_t we use $\leftarrow j_t \equiv \rightarrow(-j_t)$ and $\uparrow i_t \equiv \downarrow(-i_t)$, respectively. This means that if n is a small number, and if the tuples (i_t,j_t) are close to (0,0), for t = 1,2,...,n, then the local operator f can be computed on PBS within constant time. The following examples of local operators demonstrate this:

(i) Smoothing: Let $\sigma(X)(i,j):=X_{i,j+1}+X_{i-1,j}+X_{i,j-1}+X_{i+1,j}$, for $X\in\mathcal{B}$. Then, let

$$\underline{\text{smooth}}(X)(i,j) := \underline{\text{if}}\ (X_{ij}=1\ \underline{\text{and}}\ \sigma(X)(i,j)\geq 2)\ \underline{\text{or}}\ (X_{ij}=0\ \underline{\text{and}}\ \sigma(X)(i,j)\geq 3)\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ 0\ \underline{\text{fi}}.$$

This local operation $\underline{\text{smooth}}: \mathcal{B} \to \mathcal{B}$ has the environment $[(0,0), (0,-1), (0,+1), (-1,0), (+1,0)]$ and the Boolean function

$$x_1[\bar{x}_2 x_5 (x_3 \oplus x_4) \vee x_2 \bar{x}_5 (x_3 \oplus x_4) \vee x_2 \bar{x}_3 \bar{x}_4 \bar{x}_5 \vee \bar{x}_2 x_3 x_4 \bar{x}_5]$$
$$\vee x_4 x_5 (x_2 \oplus x_3) \vee x_2 x_3 (x_4 \oplus x_5) \vee x_2 x_3 x_4 x_5 .$$

(ii) Shrinking: We use the function σ defined above. Then, let

$$\underline{\text{shrink}}(X)(i,j) := \underline{\text{if}}\ X_{ij}=1\ \underline{\text{and}}\ \sigma(X)(i,j)=4\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ 0\ \underline{\text{fi}}.$$

This local operation $\underline{\text{shrink}}: \mathcal{B} \to \mathcal{B}$ has the same environment as smooth and the Boolean function

$$x_1 x_2 x_3 x_4 x_5 .$$

(iii) Expanding: for

$$\underline{\text{expand}}(X)(i,j) := \underline{\text{if}}\ X_{ij}=1\ \underline{\text{or}}\ (X_{ij}=0\ \underline{\text{and}}\ \sigma(X)(i,j)\geq 1)\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ 0\ \underline{\text{fi}}$$

we have the same environment as for smooth and the Boolean function

$$x_1 \vee \bar{x}_1 (x_2 \vee x_3 \vee x_4 \vee x_5).$$

Thus we see that smooth, shrink, expand are functions in the class PBS-TIME(1).

4. Some recognition problems

In this section we will consider the recognition of rectangles, squares, and isosceles right triangles.

4.1 Rectangles

Let RECT$\subseteq\mathcal{B}$ be the set of all images which contain a single solid rectangle of 1's (with sides parallel to the sides of the image) on a background of 0's. Let rect:$\mathcal{B}\to V$ be the characteristic function of RECT. Because in the images in RECT no noise is present the best approach for computing rect seems to be the consideration of "significant points" in an input image $X\in\mathcal{B}$. This provides a fast method for information reduction.

We will compute rect by counting the local property "corner." Such an approach can be found in [8] using perceptrons as models for computation.

Proposition 1. The set RECT can be recognized on PBS within time O(log N).

Proof: We use the patterns

P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8

In a first step of the algorithm, for n=1,2,...,8 write 1 at the corner point p of a binary image Y_n iff

(or a 90° rotation of this) is the place of the observed pattern P_n in the input image X; at all other points of Y_n write 0. An example of this operation is shown in Figure 1. This first step is a sequence of 8 local operators and can be done within constant time.

The input image X belongs to RECT iff each of the images Y_1, Y_2, Y_3, Y_4 has exactly one point with value 1, and the images Y_5, Y_6, Y_7, Y_8 are identically 0 at all points.

In the second step of the algorithm, let $Y := Y_5 \vee Y_6 \vee Y_7 \vee Y_8$. If $Y \equiv 0$ then go to step 3; otherwise STOP with "X is not in RECT." According to our algorithm for zero (Section 2), this step may be performed within time O(log N).

In the third step of the algorithm, we check if all of the images Y_1, Y_2, Y_3, Y_4 have exactly one point with value 1. If this is true X belongs to RECT, otherwise not.

The open question for the algorithm just described is whether there exists a PBS program which recognizes if an input image X has exactly one point with value 1 within time O(log N). Let

one(X) := if X has exactly one point with value 1
then 1 else 0 fi.

The following program computes the function one: $B \to V$ within O(log N) steps:

input image X

```
0 0 0 0
1 0 1 0
1 1 1 0
0 1 0 0
```

```
0 0 0 0      0 0 0 0      0 0 0 0      0 0 0 0
0 0 0 0      1 0 1 0      1 0 1 0      0 0 0 0
0 0 1 0      0 0 0 0      0 0 0 0      1 0 0 0
0 1 0 0      0 0 0 0      0 0 0 0      0 1 0 0
  Y1           Y2           Y3           Y4

0 0 0 0      0 0 0 0      0 0 0 0      0 0 0 0
0 1 0 0      0 0 0 0      0 0 0 0      0 1 0 0
0 0 0 0      0 0 0 0      0 0 0 0      0 0 0 0
0 0 0 0      1 0 0 0      0 0 1 0      0 0 0 0
  Y5           Y6           Y7           Y8
```

Figure 1. Examples of Y_i images.

```
begin vector A,B,C; index I; image X,Y,Z;
      int I=N/2; read X; image Y≡0;
      while I≠0 do
          Z:=(X↓I)∨X; Y:=Y∨((X↓I)∧X);
          X:=Z; I:=I→od;
      A:=X;
      if A=0 then print 0
      else
          if Y≠0 then print 0
          else int I=N/2; int C=0;
              while I≠0 do
                  B:=(A→I)∨A; C:=C∨((A→I)∧A);
                  A:=B; I:=I→od;
              if C≠0 then print 0
              else print 1 fi;
          fi;
      fi;
end
```

After the first while-loop of this program we put the lowest row of X into the vector register A. If in the input image X there was any point with value 1 then A≠0, otherwise we stop with output 0 after O(log N)steps. If in the input image X there was any column with more than one point with value 1 then Y≠0. The test "Y≠0?" needs O(log N) steps. If Y≠0, we stop with

output 0 after O(log N) steps. Now we know that A≠0 and in each column of the input image X there was at most one point with value 1. After the second while-loop we check the register C. If in A at the starting point of the while-loop there was more than one bit position with value 1 then C≠0, and we stop with output 0 after O(log N) steps. If C=0, then in the input image X there was exactly one column with exactly one point with value 1. We stop with output 1 after O(log N) steps. □

4.2 Squares

Let SQUA$\subseteq\mathcal{B}$ be the set of all images which contain a single solid square of 1's (with sides parallel to the sides of the image) on a background of 0's. Let squa:$\mathcal{B}\to\mathcal{B}$ be the characteristic function of SQUA.

Proposition 2. The set SQUA can be recognized on PBS within time O(log N).

Proof: According to the relation SQUA$\subseteq$RECT and to Proposition 1 we are able to assume that the input image X is in RECT already. The problem is to decide between squares and nonsquare rectangles.

For the computation of squa we use the images Y_1, Y_2, Y_3, Y_4 as defined in Section 4.1. Let $X := Y_1 \vee Y_2 \vee Y_3 \vee Y_4$; then in X exactly the four corners of the input rectangle are the points with value 1. In the image Y_1 there is a 1 in the position of the lower right corner of the rectangle. We spread this 1 along a diagonal path toward the upper left corner of the image. In the image Y_3 there is a 1 in the position of the upper left corner of the rectangle. We check if this 1 is on that diagonal path or not. In the positive case we know that the rectangle is a square:

```
begin image Y1,Y3; index I;
      read Y1,Y3; int I=1;
      while I≠0 do
          Y1:=Y1∨((Y1←I)↑I); I:=I←od;
      Y1:=Y1∧Y3;
      if Y1≠0 then print 1 else print 0 fi
end
```

This program runs within time O(log N). Note that the 1 in the index register I is shifted out of the log N+1 bit positions of I to the left after log N+1 runs through the while-loop. □

Propositions 1 and 2 show that the sets RECT and SQUA can be decided within the same asymptotic time as is needed when using UPCA's as models for computation [9].

4.3 Isosceles right triangles

Let IRTR$\subseteq\mathcal{B}$ be the set of all images which contain a single solid isosceles right triangle of 1's (with two sides parallel to the sides of the image) on a background of 0's. Figure 2 shows some examples of such triangles. Let irtr:$\mathcal{B}\to\mathcal{B}$ be the characteristic function of IRTR.

Proposition 3. The set IRTR can be recognized on PBS within time O(log N).

Proof: An image X is in IRTR iff for i=1,2,3,4, one of the Y_i-images marks the hypotenuse; two of the Y_i-images mark the corner points of the hypotenuse; one of the Y_i-images marks the top opposite the hypotenuse; in Y_1,Y_2,Y_3,Y_4 there are no other points with value 1; for j=5,6,7,8, three of the Y_j-images are identically 0, while the fourth Y_j-image marks the points below or above the hypotenuse; and in this fourth Y_j-image there is no further point with value 1 (if the input image is the smallest triangle consisting of a single 1 only the fourth Y_j-image is identically 0).

Figure 3 provides a visualization of this property. According to this property, the following algorithm is a decision procedure for IRTR:

Step 1. Compute the images $Y_1,Y_2,\ldots,Y_8$ in O(1) steps.

Step 2. Check that three images of Y_5,Y_6,Y_7,Y_8 are identically 0; let Z be the fourth one. This may be performed within time O(log N).

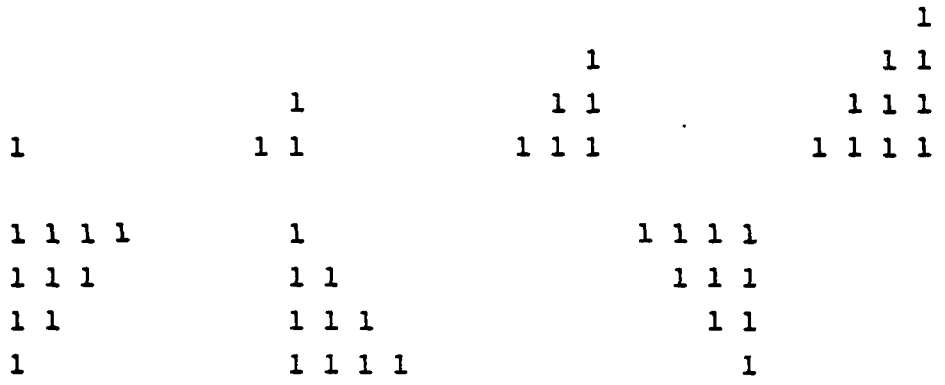


Figure 2. Examples of images in IRTR.

Input image X

1 1 1 1
1 1 1 0
1 1 0 0
1 0 0 0

0 0 0 1	0 0 0 1	1 0 0 0	0 0 0 0
0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0
0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0
Y_1	Y_2	Y_3	Y_4

0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0
0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0
Y_5	Y_6	Y_7	Y_8

Figure 3. Y_i-images for an IRTR image.

Step 3. Check that in three images of Y_1,Y_2,Y_3,Y_4 there is exactly one point with value 1; let Y_1,Y_2,Y_3 be these three images. This can be done within time O(log N).

Step 4. Check that Z and Y_4 have the right connection to each other, i.e., in Y_4 there are points marked one diagonal below or above the points which are marked in Z, and in Y_4 there is one point with value 1 less than in Z. To do this, we shift Y_4 one row up or down, compare the result with Z, using ⊕, and check if in this resulting image there is exactly one point with value 1.

Step 5. Check which of Y_1,Y_2,Y_3 mark points in the same area as Y_4 (the corner points of the hypotenuse); let Y_1,Y_2 be these two images. This can be done within time O(log N), using ⊕ and zero.

Step 6. Check if the connection between the two points in Y_1 and Y_2 marks the same path as Y_4 within time O(log N):

a) Spread the points with value 1 in Y_1 and Y_2 in the diagonal direction using power-of-2 shifts.

b) Compare Y_4 with the resulting images using ∧ and zero.

Step 7. Spread the points with value 1 in Y_1 and Y_2 in the row and column direction using power-of-2 shifts. Check that the point which is marked in Y_3 is a crossing point of these rows and columns, using ∧ and zero. This can be done within time O(log N). □

Altogether, we have a decision procedure for IRTR running within time O(log N). Possibly this procedure can be simplified by a particular property of the Y_i-images, e.g., it may be possible that we have the correct answer already after Step 6. Such an analysis to find the fastest way (within the O(log N) complexity) would be necessary for an implementation of this procedure.

5. Area and perimeter

Let S be a simply connected object [1] in a binary image X, i.e., an object without any holes. There exist several ways to define the area and the perimeter of such an object. Conventionally [1], the area of S is defined as the number of points in S. Using this definition, using number(X) we already have the area of S within $O((\log N)^2)$ steps. If we define the perimeter as the number of border points (using the 4-neighborhood) of S then we can compute the perimeter as number(edge(X)) within time $O((\log N)^2)$ too.

Now, we will define the area and the perimeter of S following the definitions in [10]. Let

$$\underline{\text{area}}(X) := \frac{1}{2}\, b+i-1$$

where b is the number of border points of S and i is the number of interior points of S. (We assume here that $i>0$.) This definition of the area of S follows the classical theorem due to PICK (1899) on the area of any simple polygon (whose edges do not cross each other) whose vertices are lattice points.

For the computation of area, we have b=number(edge(X)) and i=number(edge(X)●X) after $O((\log N)^2)$ steps. For division by 2 we need a special agreement for the interpretation of the contents of vector registers. To this end, we fix the point position between the first and the second bit positions from the right in such a register. Division by 2 represents a shift by one bit

position to the right only. Using the available Boolean operations and shifts on vector registers, addition and subtraction can be performed within time O(log N) [5]. Thus we have:

Proposition 4. The function area can be computed on PBS within time $O((\log N)^2)$.

For the definition of the perimeter of S, we use the following NxN matrices P and Q:

P(i,j):=if (i,j) is a border point
    then number of border points in the 8-neighborhood of (i,j) in the horizontal or vertical direction
    else 0 fi;

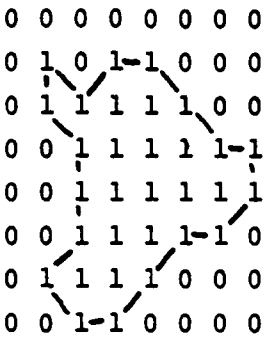
Q(i,j):=if (i,j) is a border point
    then number of border points in the 8-neighborhood of (i,j) in the diagonal direction
    else 0 fi;

for i,j = 0,1,...,N-1. Figure 4 gives a visualization of these two matrices P and Q. Then the perimeter of S is defined as

$$\underline{peri}(X) := \frac{1}{2}\left[\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} P(i,j) + \sqrt{2}\cdot\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} Q(i,j)\right].$$

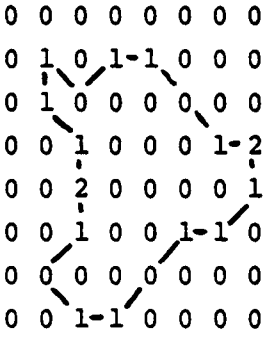
For each (i,j) the values P(i,j), Q(i,j) are in the set {0,1,2}. Therefore, we can represent P and Q by two binary images P_1,P_0 and Q_1,Q_0, respectively (cp. [6]), e.g., $P_1(i,j)P_0(i,j)$ denotes the binary representation of P(i,j). For the computation of the matrices P and Q, i.e., for the binary images P_1,P_0,Q_1,Q_0

input image X



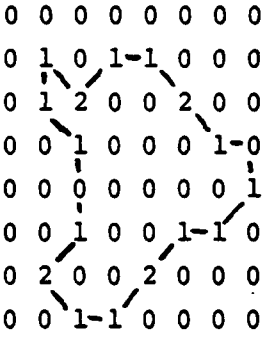
peri(X)=8+10$\sqrt{2}$

matrix P



$\Sigma\ \Sigma P(i,j)=16=2\cdot 8$
i j

matrix Q



$\Sigma\ \Sigma Q(i,j)=20=2\cdot 10$
i j

Figure 4. The matrices P and Q.

we perform four local operations on the input image X only (cp. [10]) within constant time. Using the function number and the addition on vector registers we get the values $\frac{1}{2}\Sigma\Sigma P(i,j)$ and $\frac{1}{2}\Sigma\Sigma Q(i,j)$ within time $O((\log N)^2)$. For multiplication by $\sqrt{2}$ we use the convention that the first bit position of a vector register (i.e., on the right end) indicates if the number in this vector register is a multiple of $\sqrt{2}$ or not. For example

0	0	0	1	0	0	0	0	8
0	0	0	1	0	1	0	1	$10\sqrt{2}$

is the final output for the object S in Figure 4. Thus, we have:

Proposition 5. The function peri(X)=$a+b\sqrt{2}$ can be computed on PBS within time $O((\log N)^2)$.

6. Conclusions

We have defined a model for parallel processing (PBS) which formalizes Boolean operations and shifts on binary images (Boolean matrices). It was shown that power-of-two shifts are very useful for fast programs using a divide-and-conquer approach. The matrix registers of the model used for computation represent bit planes of a possible hardware implementation of this system. Using several bit planes, grayscale pictures can be encoded in this system. In this sense the algorithms demonstrated on binary images (local operations, recognition of geometric objects, area, perimeter) are only small examples of problems which can be solved using the PBS.

References

[1] Rosenfeld, A., and A. Kak. Digital Picture Processing. Academic Press, New York, 1976.

[2] Klette, R. A Parallel Computer for Digital Image Processing. EIK 15 (1979), 237-263.

[3] Schaefer, D. H. and J. P. Strong. Tse Computers. Goddard Space Flight Center Report X-943-75-14, January 1975.

[4] Milgram, D. L., A. Rosenfeld, T. Willett, and G. Tisdale. Algorithms and Hardware Technology for Image Recognition. Final Report to U.S. Army Night Vision Laboratory, Fort Belvoir, March 31, 1978.

[5] Pratt, V. R. and L. J. Stockmeyer. A Characterization of the Power of Vector Machines. J. Computer Syst. Sci. 12 (1976), 198-227.

[6] Klette, R. Fast Matrix Multiplication by Boolean RAM in Linear Storage. Lecture Notes in Computer Science 64 (1978), 308-314.

[7] Aho, A.V., J. E. Hopcroft, and J. D. Ullman. The Design and Analysis of Computer Algorithms. Addison Wesley, Reading, Mass., 1974.

[8] Minsky, M. and S. Papert. Perceptrons. MIT Press, Cambridge, Mass., 1969.

[9] Dyer, C. R. Augmented Cellular Automata for Image Analysis. Dissertation, University of Maryland, College Park, 1979.

[10] Sankar, P. V. and E. V. Krishnamurthy. On the Compactness of Subsets of Digital Pictures. University of Maryland, Computer Science Center, TR-587, September 1977.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR- 80-0078 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): PARALLEL OPERATIONS ON BINARY IMAGES	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER: TR-822
7. AUTHOR(s): Reinhard Klette	8. CONTRACT OR GRANT NUMBER(s): AFOSR 77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Maryland, Computer Vision Laboratory, Computer Science Center College Park, MD 20742	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC 20332	12. REPORT DATE: November 1979
	13. NUMBER OF PAGES: 32
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Parallel processing
Pattern recognition
Binary images

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

It is a well-known fact that parallel logical operations and shifts are useful for speeding up certain computational tasks in binary image processing. A theoretical model for computation is given using these operations as basic instructions. Some examples demonstrate the utility of such a parallel processing system for fast solutions, e.g. the recognition of rectangles, squares, and isosceles right triangles can be done within time O(log N), for input images of size triangles can be done within time O(log N), for input images of size NxN.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)